**Resumen Final Arqui**

**Arquitectura Von Neumann**

• La unidad central de procesamiento (CPU) está constituida por la unidad de control (UC) y la unidad aritmético-lógica (ALU).

• Datos e instrucciones deben introducirse en el sistema y los resultados se proporcionarán mediante componentes de entrada/salida (E/S).

• Se necesita almacenar temporalmente datos e instrucciones: Memoria Principal

**Subrutinas**

Las subrutinas fueron innovación en lenguajes de programación. Programa auto-contenido que puede invocarse desde cualquier punto de un programa, mediante instrucción CALL. Brinda economía (código usado varias veces) y modularidad (subdivisión en unidades pequeñas). Requiere pasaje de argumentos (parámetros): - Por valor (copia de una variable). - Por referencia (dirección de la variable).

**Pasaje de parámetros** a subrutinas

Vía registros: El número de registros es la principal limitación. Es importante documentar que registros se usan

Vía memoria: Se usa un área definida de memoria (RAM). Difícil de estandarizar.

Vía pila (stack): Es el método más ampliamente usado. El verdadero “pasaje de parámetros”. Independiente de memoria y registros. Hay que comprender bien cómo funciona porque la pila (stack) es usada por el u suario y por el sistema. En x86, SP apunta al último lugar usado

**Pila (stack)**

Una pila es una estructura de datos en la que el modo de acceso a sus elementos es de tipo LIFO (último en entrar, primero en salir) que permite almacenar y recuperar datos. Para el manejo de los datos se cuenta con dos operaciones básicas: **apilar (push),** que coloca un objeto en la pila, y su operación inversa, **desapilar (pop),** que retira el último elemento apilado. En cada momento sólo se tiene acceso a la parte superior de la pila, es decir, al último objeto apilado (top). La operación desapilar permite la obtención de este elemento, que es retirado de la pila permitiendo el acceso al siguiente (apilado con anterioridad), que pasa a ser el nuevo top. En Assembler, se puede acceder a la pila a través de registro **SP (stack pointer)** que apunta por defecto al tope de la pila (8000H) y se actualiza a medida que se agregan o sacan datos.

**Principales funciones de la pila**

Pasaje de parámetros entre el programa principal y una o varias subrutinas:

* VALOR: apilando registros que contienen los datos.
* REFERENCIA: apilando las direcciones efectivas de los datos.

Guardar el contexto del procesador:

* Guardar el PC: que contiene la dirección de la próxima instrucción a ejecutar.
* Guardar el estado del procesador en ese momento (flags).

**Anidamiento de múltiples funciones utilizando pila**

El comportamiento de la pila con anidamiento de procedimientos/funciones/subrutinas: Antes de llamar a la subrutina se debe apilar los parámetros a pasar y la dirección de retorno, al llamar a la subrutina de debe:

1. Salvar el estado de BP (viejo BP) BP es el puntero al área de la pila asignada al procedimiento (frame pointer).

2. Salvar estado de SP (BP=SP)

3. Reservar espacio para datos locales (opcional)

4. Salvar valores de otros registros (opcional)

5. Acceder a parámetros

6. Escribir sentencias a ejecutar

7. Retornar parámetro (opcional)

8. Regresar correctamente del procedimiento

**Interrupciones**

Una interrupción es un mecanismo mediante el cual se puede interrumpir el procesamiento normal de la CPU. Puede ser de origen interno o externo a la CPU.

Dentro del computador, cada instrucción se representa por una secuencia de bits. El procesador debe ser capaz de extraer los datos de los distintos campos de la instrucción para realizar la operación requerida.

**¿Por qué se genera una interrupción?**

* Por el resultado de una ejecución de una instrucción. Ejemplo: desbordamiento aritmético (“overflow”), división por cero.
* Por un temporizador interno del procesador. Permite al SO realizar ciertas funciones de manera regular.
* Por una operación de E/S. Ejemplo: para indicar la finalización normal de una operación.
* Por un fallo de hardware. Ejemplo: un error de paridad en la memoria, pérdida de energía.

**¿Qué se hace si se genera una interrupción? (el pic hace lo mismo)**

En casi todos los casos, implica transferir el control a otro programa (el gestor) que:

* Suspende la ejecución del programa en curso.
* Guarda el contexto actual, próxima instrucción y estado del procesador.
* Carga el PC con la dirección de comienzo de una rutina de gestión de interrupción, se inhiben otras interrupciones.
* Finalizada la rutina de gestión, restaura el estado original del procesador.
* Retorna la ejecución normal del programa interrumpido.

**Jerarquía de interrupciones**

Si hay múltiples fuentes que pueden solicitar una interrupción se establece cuáles son más importantes.

* NO ENMASCARABLES, las que NO pueden ignorarse. Indican eventos peligrosos o de alta prioridad.
* ENMASCARABLES, pueden ser ignoradas. Con instrucciones podemos inhibir la posible solicitud.

**Tipos de interrupciones**

**Por HARDWARE**

* Son las generadas por un dispositivo de E/S.
* Son las “verdaderas” interrupciones.
* No están relacionadas con el proceso en ejecución en ese momento. El sistema de cómputo tiene que manejar estos eventos externos “no planeados” o “asincrónicos”.

**TRAPS/EXCEPCIONES**

Interrupciones por hardware creadas por el procesador en respuesta a ciertos eventos como:

* Condiciones excepcionales: overflow en ALU de punto flotante.
* Falla de programa: tratar de ejecutar una instrucción no definida.
* Fallas de hardware: error de paridad de memoria.
* Accesos no alineados o a zonas de memoria protegidos.

**Por SOFTWARE**

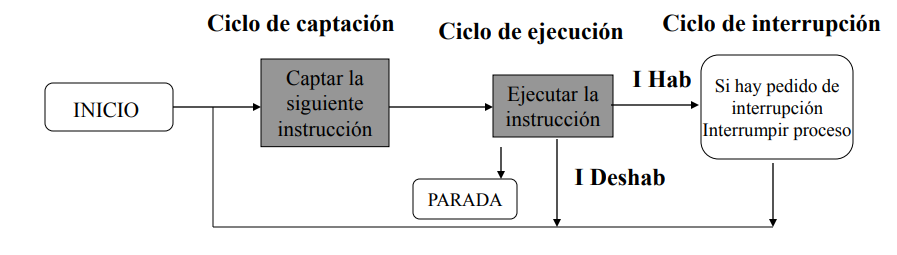
Muchos procesadores tienen instrucciones explícitas que afectan al procesador de la misma manera que las interrupciones por hardware.

* Generalmente usadas para hacer llamadas a funciones del SO. Esta característica permite que las subrutinas del sistema se carguen en cualquier lugar.
* No requieren conocer la dirección de la rutina en tiempo de ejecución.

**Las interrupciones y el ciclo de instrucción (captación, ejecución, gestión de interrupciones)**

El procesador comprueba si se ha generado alguna interrupción, indicada por una señal de interrupción. Si no hay señales de interrupción pendientes el procesador continua con el ciclo de captación y accede a la siguiente instrucción del programa en curso. Si hay alguna interrupción pendiente suspende la ejecución del programa en curso y guarda su contexto, esto significa almacenar la dirección de la siguiente instrucción a ejecutar y carga el PC con la dirección de comienzo de una rutina de gestión interrupción. A continuación, el procesador prosigue con el ciclo de captación y accede a la primera instrucción del programa de gestión de interrupción que dará servicio a esta.

Finalizada la rutina de gestión, el procesador retoma la ejecución del programa del usuario en el punto de interrupción.



**Interrupciones Múltiples**

En muchos casos, es posible que se produzca una interrupción, mientras la CPU está ejecutando otra. Para solucionar este problema se pueden seguir dos alternativas:

* **Interrupciones inhabilitadas**, se desactivan las interrupciones, mientras se está procesando una. De este modo las interrupciones entrantes quedan inhabilitadas, la CPU ignora la señal de petición de interrupción, y estas se mantienen pendientes, así cuando se activen nuevamente las interrupciones serán atendidas.
* **Definir prioridades** para las interrupciones y permitir que una interrupción de prioridad más alta pueda interrumpir a un gestor de interrupciones de prioridad menor.

**Reconocimiento de Interrupciones**

**INTERRUPCIONES MULTINIVEL**

* Cada dispositivo que puede provocar interrupción tiene una entrada física de interrupción conectada a la CPU.
* Es muy sencillo, pero muy caro.

**LINEA DE INTERRUPCIÓN ÚNICA**

* Una sola entrada física de pedido de interrupción a la que están conectados todos los dispositivos.
* Se debe “preguntar” a cada dispositivo si ha producido el pedido de interrupción (técnica Polling/encuesta)

**INTERRUPCIONES VECTORIZADAS** (vector de interrupciones)

* El dispositivo que quiere interrumpir además de la señal de pedido de interrupción, debe colocar en el bus de datos un identificador (vector). Lo coloca el periférico directamente o el controlador de interrupciones que se ocupa de todo.

**PIC Controlador de interrupciones**

El controlador de interrupciones programables es un dispositivo interno con 4 lineas de interrupción por hardware, donde se conectan dispositivos que pueden interrumpir a la CPU. De esta forma el PIC permite utilizar varios dispositivos, multiplexando los pedidos de todos ellos a la única línea de interrupción del procesador.

Registros internos

* EOI: para comandos. Para fin de interrupción escribir 20H.
* IMR: máscara de interrupción. Enmascara con “1”.
* IRR: petición de interrupción. Indica con bit en 1.
* ISR: interrupción en servicio. Indica con bit en 1.
* INT0 … INT7: 8 registros, donde carga el valor del vector de interrupción correspondiente

**¿Cómo funciona?**

El controlador de interrupciones puede manejar hasta ocho peticiones de interrupción independientes al mismo tiempo, numeradas de la 0 (INT0) a la 7 (INT7), de las cuales seleccionará una única para presentarla a la entrada de interrupción INT de la CPU.

Si más de una petición de interrupción se producen exactamente al mismo tiempo entonces el PIC las pasa a la CPU en un orden de prioridad, donde la petición por la entrada 0 tiene la prioridad más alta y la de la 7 la menor.

En el ciclo de interrupción, el procesador comprueba si se ha generado alguna interrupción. Si no hay señal, el procesador continúa con el ciclo de captación (capta la instrucción siguiente) y si hay alguna interrupción pendiente:

• Suspende la ejecución del programa en curso y guarda su contexto (dirección de la siguiente instrucción a ejecutar -el contenido del PC- y el estado del procesador)

• Carga el PC con la dirección de comienzo de una rutina de gestión de interrupción

• El procesador prosigue con el ciclo de captación y accede a la primera instrucción del programa de gestión de interrupción que dará servicio a la interrupción.

• Al completarse la rutina de gestión de interrupción, el procesador prosigue la ejecución del programa de usuario en el punto que se interrumpió

**Entrada/Salida**

El módulo de E/S es responsable de controlar a uno o más dispositivos externos y de intercambiar datos entre estos dispositivos y la memoria principal o alguno de los registros de la CPU. Este módulo consta de dos interfaces: una interna al ordenador (a la CPU y a la memoria) y otra externa (a los periféricos).

**Estructura de un módulo de E/S**

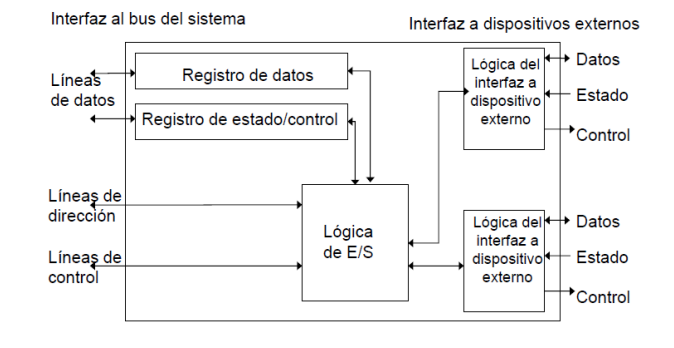
El módulo se conecta al resto del computador a través de un conjunto de líneas (como las líneas del bus del sistema).

• Registro de datos: Los datos que se transfieren a y desde el módulo se almacenan temporalmente en uno o más registros de datos.

• Registro de estado/control: Además puede haber uno o más registros de estado que proporcionan información del estado presente. Un registro de estado también puede funcionar como un registro de control, para recibir información de control del procesador

La lógica que hay en el módulo interactúa con el procesador a través de una serie de líneas de control. Son las que usa el procesador para proporcionar las órdenes al módulo de E/S. Algunas de las líneas de control pueden ser utilizadas por el módulo de E/S (ej: para las señales de arbitraje y estado). El módulo también debe ser capaz de reconocer y generar las direcciones asociadas a los dispositivos que controla. Cada módulo de E/S tiene una dirección única o, si controla más de un dispositivo externo, un conjunto de direcciones. Por último, el módulo de E/S posee la lógica específica para la interfaz con cada uno de los dispositivos que controla.

**Diagrama en bloques de un Módulo de E/S**



**Funciones de un módulo de E/S**

* Control, temporización y comunicación de uno o más dispositivos externos.
* Comunicación con la CPU (registros) y Memoria.
* Controlar las transferencias de datos entre CPU y el periférico (convertir formatos, adaptar velocidades).
* Informar a la CPU del estado del periférico.
* Almacenamiento temporal (buffering) de datos.
* Detecciones de errores.

**¿Cómo funciona un módulo de E/S?**

El funcionamiento de un módulo de E/S permite que el procesador vea a una amplia gama de dispositivos de una forma simplificada. El módulo de E/S debe ocultar los detalles de temporización, formatos y electromecánicos de los dispositivos externos para que el procesador pueda funcionar únicamente en términos de órdenes de lectura y escritura. Realiza la interfaz entre el procesador y la memoria (bus) y los periféricos.

**E/S Mapeada en memoria:** Se necesita solo una línea de lectura y solo una línea de escritura en el bus. Existe un único espacio de direcciones para las posiciones de memoria y los dispositivos de E/S.

**E/S Aislada:** Puesto que el espacio de direcciones de E/S está aislado del de memoria, este se conoce como E/S aislada. Los puertos de E/S solo son accesibles mediante una orden especifica de E/S, que activa las ordenes de E/S del bus. La ventaja es que se puede utilizar un amplio repertorio de instrucciones permitiendo una programación más eficiente.

Tanto las E/S mapeada en memoria y aislada se usan comúnmente.

**Tecnicas de gestión de E/S**

**E/S programada con espera de respuesta**

Cuando el procesador está ejecutando un programa y encuentra una instrucción relacionada con una E/S, ejecuta dicha instrucción mandando una orden al módulo de E/S apropiado. Con E/S programada el módulo de E/S realiza la acción solicitada, y después activa los bits apropiados en el registro de estado de E/S. El módulo de E/S no realiza ninguna otra acción para avisar al procesador (no interrumpe). Cuando el procesador envía una orden al módulo de E/S debe esperar hasta que la operación de E/S concluya. Si el procesador es más rápido que el módulo de E/S, el procesador desperdicia este tiempo . De esta forma el procesador es responsable de comprobar periódicamente el estado del módulo de E/S hasta que encuentra que la operación ha terminado.

**Características:**

Intercambio de datos entre la CPU y el módulo.

La CPU tiene control directo sobre la operación de E/S.

La CPU espera que el módulo E/S termine la operación.

Por lo tanto, la CPU permanece ociosa durante un periodo de tiempo.

**Ordenes de E/S**

Hay 4 tipos de ordenes de E/S que puede recibir un módulo de E/S cuando es direccionado por el procesador:

Control: Se utiliza para activar el periférico e indicarle que hacer.

Test: Se utiliza para comprobar diversas condiciones de estado asociadas con el módulo de E/S y sus periféricos . Ejemplo : comprobar si el periférico está conectado.

Lectura/Escritura: Transfiere datos desde o hacia el dispositivo por el bus de datos.

**E/S mediante interrupciones**

Tras enviar una orden de E/S a un módulo, el procesador continúa realizando algún trabajo útil. Después el módulo de E/S interrumpirá al procesador para solicitar su servicio cuando esté preparado para intercambiar datos con él. El procesador ejecuta entonces la transferencia de datos como antes, y después continúa con el procesamiento previo.

Características

* La CPU no tiene que esperar la finalización de la tarea de E/S, puede seguir procesando.
* No se repite la comprobación de los estados de los módulos.
* El módulo envía un pedido de interrupción a la CPU cuando está listo nuevamente

**Cuestiones de diseño**

En la implementación de las E/S mediante interrupciones aparecen dos cuestiones . Primero como determina el procesador que dispositivo ha provocado la interrupción y segundo, si se han producido varias interrupciones, como decide el procesador la que debe atender.

Considerando la **identificación del dispositivo** las técnicas que se utilizan comúnmente son:

Diferentes líneas para cada módulo

• No resulta practico dedicar mas de unas pocas líneas del procesador a ser líneas de interrupción porque si se utilizan varias líneas es probable que a cada una se conecten varios módulos de E/S.

• Limita el número de dispositivos

Consulta software (Poll o encuesta)

• Ocurrido un pedido de interrupción la CPU consulta a cada módulo para determinar quién fue el demandante.

• Resulta lento.

Conexión en cadena (daisy chain) “hard poll”

• La línea de reconocimiento de interrupción se conecta encadenando los módulos, la línea de pedido es compartida.

• Una vez enviada la confirmación de parte de la CPU el módulo responderá colocando un vector (palabra), en el bus, que lo identifica.

• La CPU emplea el vector como puntero para acceder a la rutina de servicio.

Considerando **interrupciones múltiples**

* Todas las líneas de interrupción tienen un orden de prioridad
* Las líneas con más prioridad pueden interrumpir a las líneas con menor prioridad.
* Si existe un maestro del bus, solo él puede interrumpir.

**Inconvenientes de la E/S programada y con interrupciones**

Las operaciones de E/S mediante interrupciones son más efectivas que las programadas, pero ambas necesitan la intervención directa de la CPU por lo que:

• La velocidad de transferencia es limitada por la velocidad del procesador.

• La CPU permanece ocupada mucho tiempo durante la operación.

**DMA (Direct Access Memory)**

El controlador de DMA es un dispositivo capaz de controlar una transferencia de datos entre un periférico y memoria sin intervención de la CPU, aumentando el rendimiento de aplicaciones. Permite a dispositivos de diferentes velocidades comunicarse sin someter a la CPU a una carga masiva de interrupciones. Si el volumen a transferir es grande, esta técnica es la más eficiente.

**Funcionamiento del DMA**

Cuando el procesador desea leer o escribir un bloque de datos, envía una orden al módulo de DMA con la información:

• Si se solicita una lectura o una escritura, usando la línea de control

• La dirección del dispositivo de E/S en cuestión, usando la línea de datos

• La posición inicial de memoria a partir de donde se lee o escribe, usando la línea de datos y almacenada por el DMA en su registro de direcciones

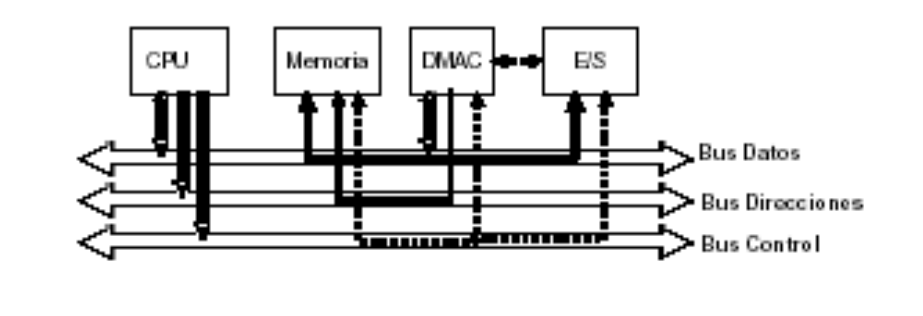
• El número de palabras a leer o escribir, usando la línea de datos y almacenando en el registro de cuenta de datos

Después el procesador continúa con otro trabajo, habiendo delegado la operación de E/S al módulo de DMA. El DMA transfiere el bloque completo de datos, palabra a palabra, directamente desde o hacia memoria sin que pase por el procesador. Cuando la transferencia terminó, el módulo de DMA envía una señal de interrupción al procesador. Así el procesador solo interviene al comienzo y al final de la transferencia.

Nota: No es una interrupción, el procesador no guarda el contexto, sino que espera durante un ciclo de bus. Eso hace que el procesador sea más lento ejecutando programas, aunque para una transferencia de E/S de varias palabras el DMA es mucho más eficiente que la E/S mediante interrupciones o programada.

**DMAC (Direct Access Memory Controller)**

El controlador de DMA (DMAC) debe actuar como maestro del bus durante la transferencia DMA y debe ser capaz de: solicitar el uso del bus mediante las señales y la lógica de arbitraje necesarias (la CPU liberará el bus cuando no lo necesite o si es forzado por el DMAC). Como también de especificar la dirección de memoria sobre la q se realiza la transferencia y generar las señales de control del bus (tipo de operación –L/E-) y señales de sincronización de trasferencia.



**Etapas de transferencia DMA**

INICIALIZACIÓN DE LA TRANSFERENCIA: La CPU debe enviar a la interfaz del periférico y al DMAC los parámetros de la transferencia. Después de la inicialización la CPU retorna a sus tareas y ya no se preocupa más de la evolución de la transferencia.

REALIZACIÓN DE LA TRANSFERENCIA: Cuando el periférico está listo para realizar la transferencia se lo indica al DMAC, este pide control del bus y se realiza la transferencia entre el periférico y la memoria. Después de cada transferencia se actualizan los registros DMAC (número de bytes a transferir y dirección de memoria).

FINALIZACIÓN DE LA TRANSFERENCIA: El DMAC libera el bus y devuelve el control a la CPU. El DMAC suele activar una señal de interrupción para indicar a la CPU la finalización de las operaciones de E/S solicitada.

**Tipos de transferencia**

Si el DMAC toma el control del bus durante los intervalos de tiempo en los que la CPU no hace uso del mismo, el rendimiento del sistema no sufrirá degradación alguna.

DMA modo ráfaga (burst): el DMAC solicita el control del bus a la CPU. Cuando la CPU concede el bus el DMAC no lo libera hasta haber finalizado la transferencia del bloque de datos completo.

* Ventaja: transferencia rápida.
* Desventaja: puede degradar el rendimiento del sistema.

DMA modo robo de ciclo (cycle-stealling): el DMAC solicita el control del bus a la CPU. Cuando la CPU concede el bus al DMAC se realiza la transferencia de una palabra y luego el DMAC libera el bus. El DMAC solicita el control del bus tantas veces como sea necesario hasta finalizar la transferencia del bloque completo.

* Ventaja: No se degrada el rendimiento del sistema.
* Desventaja: la transferencia tarda más tiempo.

• Si bien el trabajo de la CPU es lento, no será tanto como si ella realizara la transferencia. Por lo tanto, para transferencia de E/S de múltiples palabras, es la técnica más eficiente.

**Problema que puede haber con DMA**

Se puede degradar el rendimiento de la CPU si el DMAC hace uso intensivo del bus

• Si el bus está ocupado en una transferencia DMA, la CPU no puede acceder a memoria para leer instrucc. / datos

El problema se reduce con el uso de memoria cache

• La mayor parte del tiempo, la CPU lee instruc. de la cache, por lo que no necesita usar el bus de memoria.

• El DMAC puede aprovechar estos intervalos en los que la CPU está leyendo instrucciones de la cache (y por tanto no usa el bus de memoria) para realizar las transferencias.

**Canales de E/S**

Representan una extensión al concepto de DMA

Tienen la habilidad de ejecutar instrucciones de E/S y completo control de la transferencia de datos por lo tanto la CPU no ejecuta instrucciones de E/S

Programa almacenado en memoria principal .

La CPU inicia la transferencia de E/S, ordena ejecutar el programa que está en memoria

El programa especifica dispositivos, áreas de memoria a usar, prioridades y acciones ante errores.

**Tipos de canales de E/S**

Selector :

• Controla varios dispositivos de alta velocidad y uno por vez, por lo tanto, el canal se dedica para la transferencia de datos de ese dispositivo. El canal selecciona un dispositivo y efectúa la transferencia.

• Los dispositivos son manejados por un controlador o módulo de E/S.

Multiplexor:

• Puede manejar E/S con varios dispositivos a la vez.

• Multiplexor de bytes: Acepta y transmite caracteres.

• Multiplexor de bloques: Intercala bloques de datos desde distintos dispositivos.

**Segmentación de Cauce (pipeline)**

La segmentación de cauce, también denominada pipeline, es una técnica empleada en el diseño de procesadores, basada en la división de la ejecución en etapas, consiguiendo así que una instrucción empiece a ejecutarse antes de que hayan terminado las anteriores y, por lo tanto, que haya varias instrucciones procesándose simultáneamente.

**Características**

La segmentación es una técnica de mejora de prestaciones a nivel de diseño de hardware. Es invisible al programador.

Necesidad de uniformizar las etapas: Al tiempo de la más lenta y al número de etapas máximo.

El diseño de procesadores segmentados tiene gran dependencia del repertorio de instrucciones (RISC).

Incrementa la productividad, pero no reduce el tiempo de ejecución de la instrucción.

**Tareas a realizar por ciclo**

Búsqueda (F, Fetch) MI

* Se accede a memoria por la instrucción
* Se incrementa el PC

Decodificación (D, Decode) BR

* Se decodifica la instrucción, obteniendo la operación a realizar en la ruta de datos.
* Se accede al banco de registros por los operandos (si es necesario).
* Se calcula el valor del operando inmediato con extensión de signo si hace falta)

Ejecución (X, Execute) ALU

* Se ejecuta la operación en la ALU.

Acceso a memoria (M, Memory Access) MD

* Si se requiere un acceso a memoria, se accede.

Almacenamiento (W, Writeback) BR

* Si se requiere volcar un resultado a un registro, se accede al banco de registros.

Para conseguir una mayor aceleración, el cauce debe tener más etapas. Todas las etapas deben tardar lo mismo y el tiempo del ciclo será el de la etapa más lenta, pero los ciclos de instrucción serán menos lo que provoca un aumento del rendimiento.

**Prestaciones del cauce segmentado**

Teórica: El máximo rendimiento es completar una instrucción con cada ciclo de reloj.

Si K es el número de etapas del cauce ⇒ Vel. procesador segmentado = Vel. secuencial x K

El incremento potencial de la segmentación del cauce es proporcional al número de etapas del cauce.

El tiempo de ciclo T de un cauce de instrucciones es el tiempo necesario para que un conjunto de instrucciones avance a una etapa a través del cauce.

T = max [Ti] + d = Tm + d i, 1<= i <= k donde:

Tm = máximo retardo de etapa k = numero de etapas del cauce

d = retardo de tiempo de un registro.

**Problemas**

•No todas las instrucciones necesitan todas las etapas.

•No todas las etapas pueden ser manejadas en paralelo.

•No se tienen en cuenta los saltos de control.

**Atascos de un cauce (stall)**

Son aquellas situaciones que impiden a la siguiente instrucción que se ejecute en el ciclo que corresponde.

Tipos de riesgos: Estructurales, por dependencia de datos y de control

Soluciones :

**Riesgos estructurales:** Dos instrucciones necesitan utilizar el mismo recurso de hardware en el mismo ciclo. Es decir, dos accesos simultáneos a memoria, al banco de registros o a la ALU.

Las posibles soluciones que encontramos son:

* Replicación de unidades funcionales.
* Caches diferenciadas de datos e instrucciones.
* Más de un puerto de acceso a memoria o al banco de registros.

**Riesgos por dependencia de datos:** Condición en la que los operandos fuente o destino de una instrucción no están disponibles en el momento en que se necesitan en una etapa determinada del cauce. Ocurren cuando dos instrucciones se comunican por medio de un dato (Ejemplo: uno lo produce y otro lo usa).

Las posibles soluciones que encontramos son:

A nivel de hardware:

* Adelantamiento de operando (forwarding): si el dato necesario está disponible a la salida de la ALU (Ei) se lleva a la entrada de la etapa correspondiente (Ei+1) sin esperar la escritura (Mi o Wi). Para esta posible solución se necesitan registros extras. Fácil de implementar si se identifican todos los adelantamientos y se comunican a los registros de segmentación correspondientes.

A nivel software:

* Instrucciones NOP o reordenación de código, con esto generamos retardo.
* Reordenación de las instrucciones, ejecución “fuera de orden”.

**Tipos de dependencias de datos**

* LECTURA DESPUÉS DE ESCRITURA (RAW, dependencia verdadera) una instrucción genera un dato que lee otra posterior.
* ESCRITURA DESPUÉS DE ESCRITURA (WAW, dependencia en salida) una instrucción escribe un dato después que otra posterior, esto solo se da si se deja que las instrucciones se adelanten unas a las otras.
* ESCRITURA DESPUÉS DE LECTURA (WAR, Antidependencia) una instrucción modifica un valor antes de que otra anterior que lo tiene que leer, lo lea. Esto, no se puede dar en nuestro cauce simple.

**Riesgos de Control**

Necesidad de tomar una decisión basada en los resultados de una instrucción, mientras las otras se están ejecutando. (Ejemplo: un salto y los dos posibles caminos). La ejecución de una instrucción depende de cómo se ejecute la otra. Es conciso aclarar que existe una penalización por salto.

Las instrucciones de salto pueden ser:

* Incondicional: penalización de salto, la dirección de destino se debe determinar lo más pronto posible dentro del cauce, para reducir la penalización.
* Condicional: introduce riesgo adicional por la dependencia entre la condición de salto y el resultado de una instrucción previa.

Las posibles soluciones que encontramos son:

Adelantar la resolución de los saltos a la etapa D:

* En ella se decodifica y se sabe que es un salto.
* Se puede evaluar la condición de salto (con restador) y se puede calcular la dirección de salto (con sumador).

**Tratamiento de saltos (preguntar cuáles son las técnicas hardware y software y a cual le pertenecen las técnicas estáticas y dinámicas)**

Uno de los mayores problemas de diseño de un cauce de instrucciones es asegurar un flujo estable de instrucciones a las etapas iniciales del cauce. El principal obstáculo: la bifurcación condicional. Las técnicas son:

* FLUJOS MULTIPLES: varios cauces (uno por cada opción de salto). Precaptan cada salto en diferentes cauces.

Desventajas:

1. Provoca retardos en el acceso al bus y a los registros.

2. Si hay múltiples saltos, se necesita un mayor número de cauces.

* PRECAPTAR EL SALTO DESTINO: se precapta la instrucción destino del salto, además de las instrucciones siguientes a la bifurcación. La instrucción se guarda hasta que se ejecute la instrucción de bifurcación. Se produce el salto, el destino ya habrá sido precaptado.
* BUFFER DE BUCLES: memoria muy rápida. Gestionada por la etapa de captación de instrucción del cauce. Comprueba el buffer antes de hacer la captación de memoria. Muy eficaz para pequeños bucles y saltos.
* SALTO RETARDADO, no hacer el salto hasta que sea necesario Siempre se ejecuta la siguiente instrucción secuencial, el salto se ejecuta después del retardo de una instrucción.
* Las instrucciones en los huecos de retardo de salto (delay-slot) se captan siempre.
* Se trata de situar instrucciones útiles (que no dependan del salto) en los huecos de retardo. Si no es posible, se utilizan instrucciones NOP.
* Requieren reordenar las instrucciones.

**Predicción de saltos**

**Técnicas estáticas** (no dependen de la historia de ejecución):

* Predecir que nunca se salta: asume que el salto no se producirá y siempre capta la siguiente instrucción.
* Predecir que siempre se salta: asume que el salto se producirá y siempre capta la instrucción destino del salto.
* Predecir según el código de operación: hay instrucciones con más probabilidades de saltar según su codop, con esta técnica la tasa de acierto puede llegar a alcanzar un 75%.

**Técnicas dinámicas** (depende la historia de ejecución):

* Conmutador saltar/no saltar: basado en la historia de las instrucciones, eficaz para los bucles.
* Tabla de historia de saltos (branch-target buffer): pequeña caché asociada a la etapa de búsqueda (F): Tres campos:
  + - Dirección de una instrucción de bifurcación.
    - Información de la instrucción destino.
    - N bits de estado (historia de uso).

**Procesador CISC (Computadoras de repertorio complejo de instrucciones.)**

Características distintivas

* Conjunto de instrucciones muy amplio.
* Permite operaciones complejas entre operandos situados en la memoria o en los registros internos.

Se debe tener en cuenta los inconvenientes que presenta CISC:

* El software resulta mucho más caro que el hardware.
* El nivel del lenguaje era cada vez más complicado.
* Salto semántico

Todo esto conduce a: Repertorio de instrucciones grandes, más modos de direccionamiento y varias sentencias de HLL implementadas en el hardware.

**La finalidad del CISC:** Facilitar el trabajo del escritor en compiladores y mejorar la eficiencia de ejecución Secuencias complejas de operaciones en micro código.

**Características de la ejecución de instrucciones (CISC)**

Estudios sobre programas en HLL

**Secuenciamiento de la ejecución**: determina la organización y control de cauce.

**Operaciones:** Las sentencias de asignación predominan junto las sentencias condicionales (If, loop).

Estos resultados son instructivos para el diseñador del repertorio de instrucciones maquina por que le dicen que tipos de sentencias tienen lugar más a menudo y, por consiguiente, son implementados de una forma óptima.

**Operandos:** La mayoría de las referencias se hacen a variables globales escalares simples. Además, mas del 80% de los datos escalares eran variables locales. De este modo hay un predominio de referencias a operandos escalares y estos están muy localizados.

**Llamadas a procedimientos:** Son las operaciones que consumen más tiempo en programas en HLL compilados. Hay dos aspectos importantes:

* El número de parámetros y variables que trata un procedimiento y la profundidad de anidamiento
* El número de palabras por cada activación de un procedimiento no es muy grande.

Es poco común tener una larga secuencia ininterrumpida de llamadas a procedimientos seguida por la correspondiente secuencia de retornos.

**Amplio banco de registros (¿surge a partir de las consecuencias de las Llamadas a procedimientos?)**

Estos constituyen el dispositivo de almacenamiento más rápido disponible. El banco de registros es pequeño físicamente, se necesita una estrategia que permita que a los operandos a los que se accede con mayor frecuencia se encuentren en registros y que se minimicen las operaciones registro-memoria.

Son posibles dos aproximaciones básicas:

* Por software: Consiste en confiar al compilador la maximización del uso de los registros. El compilador intentara asignar registros a las variables que más se usen en un periodo de tiempo dado. Requiere el uso de sofisticados algoritmos de análisis de programas.
* Por Hardware: Consiste sencillamente en usar más registros, de manera que más variables pueden mantenerse en registros durante periodos de tiempo más largos.

**Ventanas de registro**: Múltiples conjuntos de pequeños registros cada uno asignado a un procedimiento distinto. Las ventanas se dividen en tres áreas

* Los registros de parámetros contienen parámetros pasados al procedimiento en curso desde el procedimiento que lo llamo y los resultados a devolver a este.
* Los registros locales se usan para variables locales, según la asignación que use el compilador.
* Los registros temporales se usan para intercambiar parámetros y resultados con el siguiente nivel más bajo.

**Variables globales**: El compilador asigna posiciones de memoria a variables declaradas como globales en un HLL y que todas las instrucciones maquina que referencien esas variables usen operandos referenciados en memoria.

**Amplio banco de registros VS Cache**

|  |  |
| --- | --- |
| Banco de registros | Cache |
| Todos los datos escalares locales. | Datos escalares locales recientemente usados |
| Variables individuales | Bloques de memoria |
| Variables globales asignadas por el compilador | Variables globales usadas recientemente |
| Restauraciones basadas en la profundidad de anidamiento | Restauraciones basadas en el algoritmo de reemplazo |
| Direccionamiento a registros | Direccionamiento a memoria |

**Optimización de registros basada en el compilador**

El objetivo del compilador es mantener en registros, en lugar de memoria, los operandos necesarios para tantos cálculos como sea posible y minimizar las operaciones de carga y almacenamiento.

• Cada ‘cantidad’ del programa candidata se asigna a un registro simbólico o virtual.

• Asignar el número ilimitado de registros simbólicos a un número fijo de registros reales.

• Registros simbólicos que no se solapan pueden compartir el registro real.

• Si se agotan los registros reales, algunas de las variables se asignan a posiciones de memoria.

**¿Por qué CISC?**

Programas más pequeños

* El programa ocupa menos memoria, pero la memoria hoy en día es muy barata.
* Un programa puede ser corto (pocas instrucciones), pero no tiene por qué ocupar menos bits en memoria.
* Mas instrucciones = codops más largos = instrucciones más largas.

Programas más rápidos

* Propensión a usar las instrucciones más sencillas.
* Unidad de control más compleja.

Con CISC es mucho más difícil optimizar el código generado para minimizar su tamaño, reducir el número de instrucciones ejecutadas y mejorar la segmentación.

**Procesador RISC (Computadoras de repertorio reducido de instrucciones)**

Características distintivas:

* Gran número de registros de uso general.
* Repertorio de instrucciones limitado y sencillo.
* Énfasis en la optimización de la segmentación de instrucciones.

Características principales

* Una instrucción por ciclo: Un ciclo de maquina se define como el tiempo que se tarda en captar dos operandos desde los registros, realizar una operación en la ALU y almacenar el resultado en un registro. Con instrucciones sencillas y de un ciclo, hay poca o ninguna necesidad de micro código, las instrucciones maquinas pueden estar cableadas.
* Operaciones registro a registro: Esta forma de diseño simplifica el diseño del repertorio de instrucciones y, por lo tanto, la unidad de control. Otra ventaja es que tal arquitectura fomenta el uso de registros ya que los operandos frecuentemente accedidos, permanecen en el almacenamiento de alta velocidad. Este énfasis en las operaciones registro a registro es único en los diseños RISC.
* Modo de direccionamiento sencillos: Casi todas las instrucciones RISC usan el direccionamiento sencillo a registros. Nuevamente, esta característica de diseño simplifica el repertorio de instrucciones y la unidad de control.
* Formato de instrucción sencillos: Generalmente solo se usa un formato o unos pocos. La longitud de las instrucciones es fija. Las posiciones de los campos son fijas. Hay varias ventajas, la definición del codop y el acceso a los operandos en registros puede tener lugar simultáneamente.

Los formatos sencillos simplifican la unidad de control. Se optimiza la captación de instrucciones.

* Diseño cableado (sin micro código).
* Formato de instrucción fijo.
* Mayor tiempo/esfuerzo de compilación.

**Conclusión**

La razón por la que se opta por los procesadores RISC, es para contrarrestar los inconvenientes del CISC. Se llegó a la conclusión a que es deseable: Usar un gran número de registros (optimizando las referencias a operandos). Esto se puede hacer de dos formas:

**Aproximación por software**

• El compilador es necesario para asignar registros.

• Asignación de registros a las variables que se usen más en un periodo de tiempo dado.

• Requiere el uso de sofisticados algoritmos de análisis de programas.

**Aproximación por hardware**

• Utilización de más registros.

• De esta manera, más variables pueden mantenerse en registros durante periodos de tiempo más largos.

**Controversia RISC y CISC**

La controversia entre la elección del RISC o CISC se enfoca en los siguientes aspectos de la eficiencia en la ejecución:

Cuantitativa: Comparación del tamaño de los programas y su velocidad de ejecución.

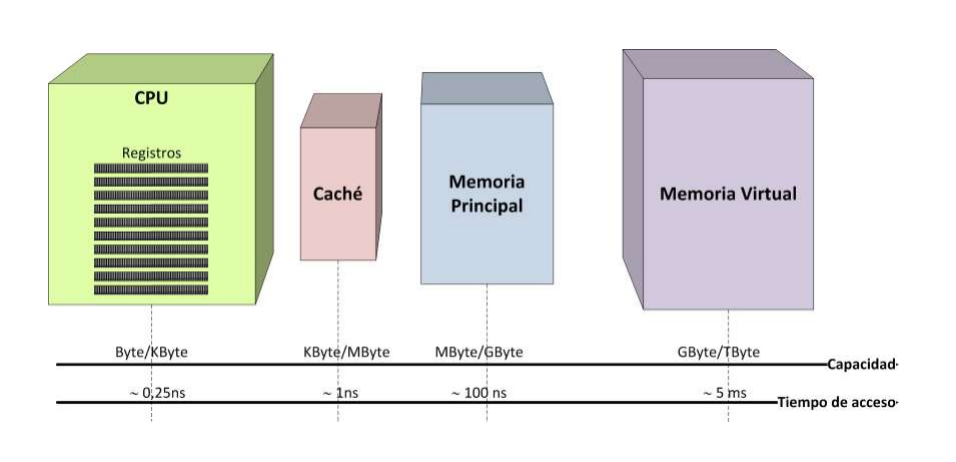
Cualitativa: Revisión de soporte de lenguajes de alto nivel y uso óptimo de los recursos VLSI (Very Large Scale Integration).

El problema de las comparaciones son que:

* No existe un par de máquinas RISC y CISC directamente comparables.
* No hay un conjunto de programas de prueba definitivo.
* Dificultad para separar los efectos del hardware de los del compilador.
* Mayoría de comparaciones con máquinas de “juguete”, no con productos comerciales.
* La mayoría de las máquinas son una mezcla de ambas.

**Jerarquía de Memoria**

La jerarquía de memoria usa diferentes tipos de memoria y de diferentes velocidades, usando una memoria caché en el medio para acelerar las funciones. Organizada en niveles que son ubicados en distintos lugares físicos, fabricados con tecnologías diferentes que se gestionan de manera independiente.



Objetivo: la velocidad del sistema deberá ser, aproximadamente, la del nivel más rápido al costo del nivel más barato. A medida que nos alejamos de la CPU, cada nivel inferior es más grande, más lento y más barato que el nivel previo (o superior) en la jerarquía.

¿Por qué funciona la jerarquía?

**Principios que sustentan la jerarquía de memoria:**

**Principio de localidad de referencias:** El uso de la memoria caché, que en palabras generales es, que cuando se hace referencia a una palabra, ella y algunas de sus vecinas se traen de memoria secundaria (lenta y grande) a la caché, que es una memoria de alta velocidad, para que el siguiente acceso a la palabra buscada se encuentre en caché. El uso de la caché se sustenta en 2 principios que tienen los programas:

* PRINCIPIO DE LOCALIDAD TEMPORAL: cuando se accede a una porción de memoria, es muy probable que en un lapso de tiempo dicha posición de memoria sea accedida nuevamente.
* PRINCIPIO DE LOCALIDAD ESPACIAL: cuando se accede a una palabra en memoria, es muy probable que el próximo acceso sea en las cercanías de la palabra anterior.

**Memoria Cache**

Es una cantidad pequeña de memoria rápida que se ubica entre la memoria principal y la CPU. Puede localizarse en un chip o en un módulo CPU.

Funcionamiento de la cache: La CPU solicita contenido de 1 dirección de memoria, la cache ¿tiene ese dato?:

* Si es así, la obtiene de la cache (rápidamente).
* Si no está, se lee el bloque que contiene esa dirección desde la memoria principal y copia en la cache.

Después, la cache entrega el dato requerido a la CPU. La cache incluye etiquetas para identificar qué bloque de la memoria principal está en cada una de sus líneas.

Conceptos básicos

* Acierto (hit): se encuentra en la caché el dato solicitado.
* Fallo (miss): no se encuentra en la caché el dato solicitado.
* Un bloque que contiene la palabra accedida se copia de la memoria principal a una línea de caché.
* Tiempo para servir un fallo: depende de la latencia y ancho de banda de la memoria principal.
* Latencia: tiempo necesario para completar un acceso a memoria.
* Ancho de banda: cantidad de información por unidad de tiempo que puede transferirse desde/hacia la memoria.
* Los fallos de caché se gestionan mediante hardware y causan que el procesador se detenga hasta que el dato esté disponible.

**Diseño de la cache**

**Tamaño**

Nos gustaría que el tamaño sea lo suficientemente pequeño como para que el coste total medio por bit sea próximo al de la memoria principal sola y lo suficientemente grande como para que el tiempo de acceso medio total fuera próximo al de la cache.

**Política de ubicación**

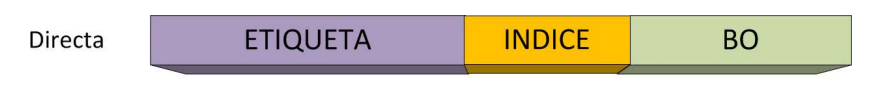
**Función de correspondencia - Ubicación de un bloque :**

La elección de la función de correspondencia determina como se organiza la cache.

**Correspondencia directa:** La técnica más simple, consiste en hacer corresponder cada bloque de memoria principal a una sola línea posible de cache. Se implementa fácilmente utilizando la dirección.

Su principal desventaja es que hay una posición concreta de cache para cada bloque dado, si un programa referencia repetidas veces a palabras de dos bloques asignados diferentemente en la misma línea, dichos bloques se estarían intercambiando continuamente en la cache y la tasa de aciertos seria baja. La dirección física tiene la forma

**INDICE** indicará la línea o el conjunto que le corresponde. **BO** representa todas las direcciones que pertenecen al bloque.



**Correspondencia asociativa:**  Supera la desventaja de la directa, permitiendo que cada bloque de memoria principal pueda cargarse en cualquier línea de la cache. La lógica de control de la cache interpreta una dirección de memoria simplemente como una etiqueta y un campo de palabra. El campo de etiqueta identifica unívocamente un bloque de memoria principal.

Hay flexibilidad para que cualquier bloque sea reemplazado cuando se va a escribir uno nuevo en la cache.

Su principal desventaja es la compleja circuitería necesaria para examinar en paralelo todas las líneas de la cache.



**Correspondencia asociativa por conjuntos:** Es una solución de compromiso que recoge lo positivo de las correspondencias directa y asociativa, sin presentar sus desventajas. En este caso la cache se divide en V conjuntos, cada uno de K líneas.

Se denomina correspondencia asociativa por conjuntos de K vías.

La etiqueta en una dirección de memoria principal es bastante larga y debe compararse con la etiqueta de cada línea de la cache.



**Política de reemplazos**

Para el caso de correspondencia directa:

• No hay elección.

• Sólo hay una posible línea para cada bloque.

• Se necesita una sustitución de esa línea (si o sí).

Para las técnicas asociativas se requieren algoritmos de sustitución. Tales algoritmos deben implementarse en hardware para conseguir velocidad. Los 4 algoritmos más comunes son:

* LRU(utilizado menos recientemente): Probablemente es el más efectivo; se sustituye el bloque que se ha mantenido en la cache por más tiempo sin haber sido referenciado. Requiere controles de tiempo. El LRU debería dar la mejor tasa de aciertos.
* FIFO(primero en entrar-primero en salir): Se sustituye aquel bloque del conjunto que ha estado más tiempo en la cache. Requiere controles de acceso.
* LFU(menos frecuentemente usado): se sustituye aquel bloque del conjunto que ha experimentado menos frecuencias. Podría implementarse asociando un contador a cada línea. Requiere controles de uso.
* Aleatoria: Consiste simplemente en tomar una línea al azar entre las posibles candidatas.

**Política de Escritura**

Política de escritura en acierto:

• ESCRITURA INMEDIATA (Write-through): se actualizan simultáneamente la posición de la caché y de la memoria principal. Se genera mucho tráfico y retrasa la escritura. Se mantiene la coherencia en todo momento. Suele combinar con la técnica “no carga en escritura” (no-write allocate).

• POST-ESCRITURA (Write-back): la información sólo se actualiza en la caché. Cuando un bloque es desalojado de la cache, se comprueba un bit y si está activo, se escribe la información de dicho bloque en memoria principal. Puede contener información errónea en algún momento. Suele combinarse con la técnica carga en escritura (write allocate).

Política de escritura en fallo:

• CARGA EN ESCRITURA (write allocate): la información se lleva de la memoria principal a la caché. Se sobrescribe en la caché. Habitual con write-back.

• NO CARGA EN ESCRITURA (no-write allocate): el bloque no se lleva a la memoria caché. Se escribe directamente en la memoria principal. Habitual con escritura inmediata (Write-through).

**Coherencia en la Cache de datos**

Para conseguir coherencia de cache, la cache de datos permite un protocolo conocido como MESI (modificada, exclusiva, compartida, no valida) diseñado para satisfacer los requisitos de coherencia de cache de un sistema multiprocesador.

* Modificada: La línea de la cache ha sido modificada (diferente de memoria principal) y esta disponible solo en la cache,
* Exclusiva: El contenido de la línea de la cache coincide con lo que se tiene en memoria principal y no está presente en ninguna otra cache.
* Compartida: La línea en la cache coincide con memoria principal y puede esta presente en otra cache.
* No valida: La línea de la cache no contiene datos válidos.

**Dos niveles de caché justificados**

La caché de dos niveles consta de:

• la caché interna, el nivel 1 (L1)

• la caché externa, el nivel 2 (L2)

Debido a la lentitud usual del bus y a los tiempos de acceso de las memorias, se obtienen bajas prestaciones. Pero si se usa una caché L2 SRAM, entonces con frecuencia la información que falta puede recuperarse fácilmente. Si la SRAM es rápida, los datos pueden accederse con cero estados de espera.

La ventaja de caché con dos niveles es que elimina la competición por la caché entre el procesador de instrucciones y la unidad de ejecución, importante en diseños que cuentan con segmentación de cauce de instrucciones (pipelining).

**Buses**

Un bus es un camino de comunicación entre dos o más dispositivos.

Al bus se conectan varios dispositivos, y cualquier señal transmitida por uno de esos dispositivos está disponible para que los otros dispositivos conectados al bus puedan acceder a ella. Si dos dispositivos transmiten durante el mismo periodo de tiempo, sus señales pueden solaparse y distorsionarse, por este motivo solo un dispositivo puede transmitir con éxito en un momento dado.

El bus que conecta a los componentes principales de la computadora (memoria, procesador, E/S) se denomina BUS DEL SISTEMA.

**Estructura del Bus**

El bus del sistema está constituido, usualmente, por entre 50 y 100 líneas. A cada línea se le asigna una función particular. Las líneas se pueden clasificar en tres grupos funcionales:

**BUS DE DATOS:** Proporcionan un camino para transmitir datos entre los módulos del sistema.

* Transmiten datos, a este nivel no existe diferencia entre “datos” e “instrucciones”.
* El ancho del bus es un factor clave a la hora de determinar las prestaciones (8, 16, 32, 64 bits).

**BUS DE DIRECCIÓN**: Identifica la fuente o destino de un “dato”, cuando el procesador desea leer una palabra de una determinada parte en la memoria.

* El ancho del bus de direcciones determina la máxima capacidad de memoria posible del sistema

**BUS DE CONTROL**: Se utiliza para controlar el acceso y el uso de las líneas de datos y de direcciones.

* Transmite información de señales de control y temporización

**Tipos de Buses**

**DEDICADOS:** Una línea dedicada está permanentemente asignada a una función o a un subconjunto físico de componentes de la computadora.

**MULTIPLEXADOS:** Uso de las mismas líneas para usos diferentes.

**Métodos de arbitraje**

**CENTRALIZADO:** Un único dispositivo de hardware denominado árbitro, es responsable de asignar tiempos en el bus. El dispositivo puede estar en un módulo separado o ser parte del procesador.

**DISTRIBUÍDO:** No existe un árbitro. En su lugar, cada módulo dispone de lógica para controlar el acceso y los módulos actúan conjuntamente para compartir el bus.

**Temporización**

**SINCRONICA:** La presencia de un evento en el bus está determinado por un reloj. El bus incluye una línea de reloj. Un único intervalo igual a 1 seguido de otro a cero se conoce como un ciclo de bus y define un intervalo de tiempo (time slot). Todos los dispositivos del bus pueden leer la línea de reloj.

**ASINCRONICA:** La presencia de un evento en el bus es consecuencia y depende de que se produzca un evento previo.

La temporización sincrónica es más fácil de implementar y comprobar. Sin embargo, es menos flexible que la temporización asincrónica. Con la temporización asincrónica, pueden compartir el bus una mezcla de dispositivos lentos y rápidos.

**Bus PCI**

Es un bus estándar de computadoras para conectar periféricos directamente en la placa base.

Popular, de ancho de banda elevado, independientemente del procesador, que se puede utilizar como bus de periféricos o bus para una arquitectura de entreplanta.

Características generales:

* Proporcionan mejores prestaciones para los subsistemas de E/S de alta velocidad.
* 32 a 64 bits.
* Fácil de implementar.
* Permite que otros buses se conecten al bus PCI.
* Utiliza temporización sincrónica y arbitraje centralizado.

**Bus SCSI**

Es una interfaz con 8, 16 o 32 líneas de datos que se utiliza para la transferencia de datos entre distintos dispositivos del bus de la computadora.

Cada dispositivo tiene dos conectores, uno de entrada y el otro de salida. Los dispositivos funcionan de forma independiente y pueden intercambiar datos entre sí como en el host.

**Procesadores superescalares**

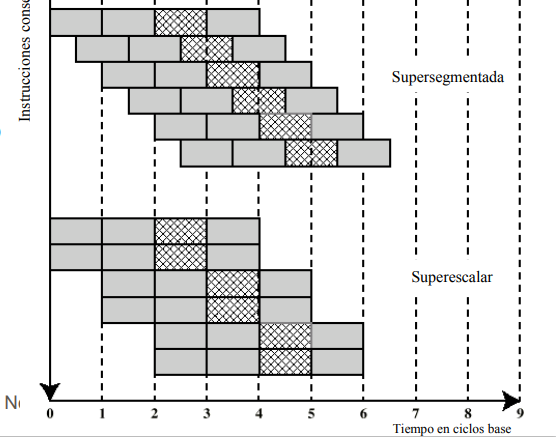
Es aquel que utiliza múltiples cauces de instrucciones independientes. Cada cauce consta de múltiples etapas, de modo que puede tratar varias instrucciones a la vez.

Capta varias instrucciones a la vez y a continuación intenta encontrar instrucciones cercanas que sea independiente entre sí y puedan ejecutarse en paralelo. Puede eliminar algunas dependencias innecesarias de registros adicionales y el renombramiento de las referencias en el código inicial.

**Procesadores supersegmentados**

Aprovecha el hecho de que muchas etapas del cauce realizan tareas que ocupan menos de la mitad de un ciclo de reloj. De este modo permite la realización de dos tareas un ciclo de reloj externo.

División de las etapas “macro” del cauce segmentado en sub-etapas más pequeñas y se transmiten los datos a la mayor velocidad del ciclo de reloj. Aumenta el grado del paralelismo.



El diagrama muestra una implementación segmentada que es capaz de ejecutar dos etapas del cauce por ciclo re reloj, es decir, cada etapa se puede dividir en dos etapas no solapadas y que cada una se ejecuta en medio ciclo de reloj.

El procesador supersegmentado se queda atrás con respecto al superescalar al comienzo del programa y en cada destino de un salto

**Paralelismo de instrucciones**

Se refiere al grado en el que las instrucciones de un programa se pueden ejecutar en paralelo. Para maximizar el paralelismo a nivel de instrucciones se puede usar una combinación de optimizaciones realizadas por el compilador y técnicas hardware.

Limitaciones:

**Dependencia de datos verdadera:** Consideremos la siguiente sentencia

Add r1,r2

Move r3,r2

La segunda instrucción se puede captar y modificar, pero no se puede ejecutar hasta que finalice la ejecución de la primera instrucción. El motivo es que la segunda instrucción necesita un dato producido por la primera instrucción.

**Dependencia relativa al procedimiento**: Las instrucciones que siguen a una bifurcación tienen una dependencia relativa al procedimiento en esa bifurcación y no pueden ejecutarse hasta que esta lo haga. Este tipo de dependencia afecta también a un cauce escalar.

**Conflictos en los recursos:** Es una pugna entre dos o más instrucciones por el mismo recurso al mismo tiempo. Presenta el mismo comportamiento que una dependencia de datos, sin embargo, los conflictos en los recursos pueden superarse duplicando estos.

**Dependencia de salida (WAW) y Antidependencia (WAR):** El paralelismo a nivel de instrucciones depende de la frecuencia de dependencias de datos verdadera y dependencias relativas al procedimiento que haya en el código.

**Políticas de emisión de instrucciones**

Podemos agrupar las políticas de emisión de instrucciones de los procesadores superescalares en las siguientes categorías.

* **Emisión y finalización en orden:** Las políticas de emisión de instrucciones más sencilla es emitir instrucciones en el orden exacto en que lo haría una ejecución secuencial y escribir los resultados en el mismo orden
* **Emisión en orden y finalización desordenada:** Se usa para mejorar la velocidad de las instrucciones que necesitan muchos ciclos. Puede haber cualquier número de instrucciones en la etapa de ejecución en un momento dado, hasta alcanzar el máximo grado de paralelismo de la máquina, ocupando todas las unidades funcionales.
* **Emisión y finalización desordenada**: Para permitirla es necesario desacoplar las etapas del cauce de decodificación y ejecución. El resultado de esta organización permite identificar instrucciones independientes que pueden introducirse en la etapa de ejecución.

**Renombre de registros**

El hardware del procesador asigna dinámicamente los registros que están asociados con los valores que necesitan las instrucciones en diversos instantes de tiempo. Cuando se crea un nuevo valor de registro se asigna un nuevo registro para este valor. Se pueden producir riesgos RAW.

**Implementación superescalar**

• Estrategias de captación simultánea de múltiples instrucciones.

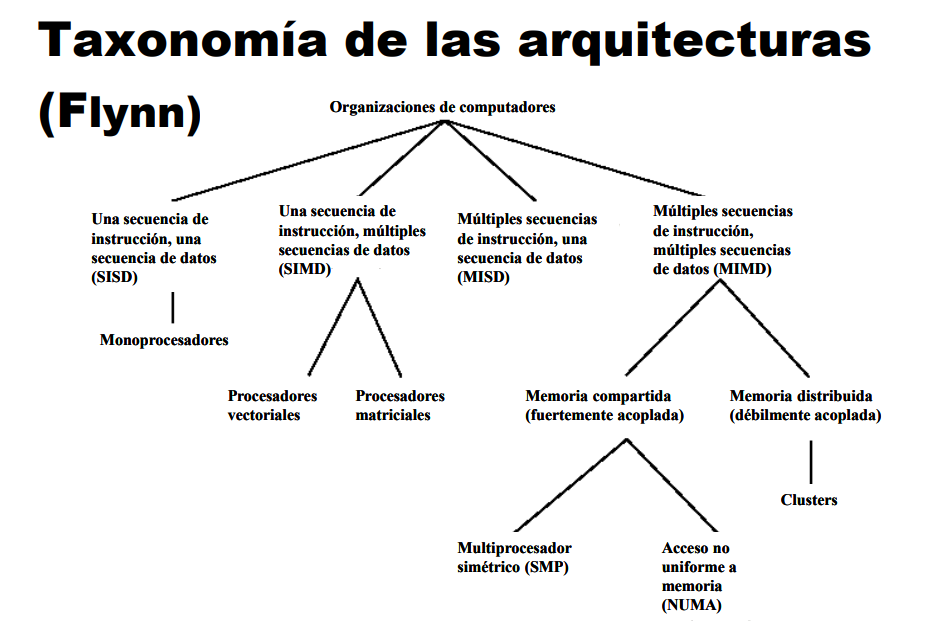
• Lógica para determinar dependencias verdaderas.

• Mecanismos para iniciar o emitir múltiples instrucciones en paralelo y para entregar el estado del procesador en un orden correcto.

• Recursos para la ejecución en paralelo de múltiples instrucciones.

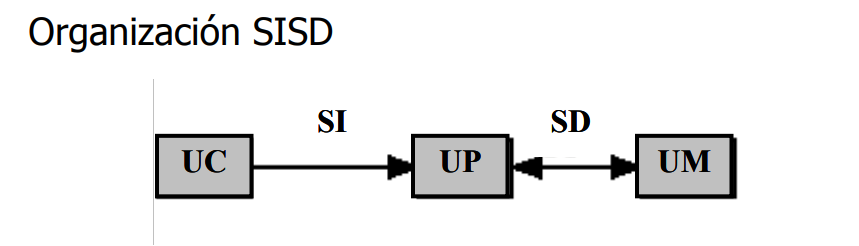
**Introducción al procesamiento paralelo (según Flynn)**

Sea cual sea el nivel de prestaciones, la demanda de máquinas de mayor rendimiento seguirá existiendo. Mejora el rendimiento de una máquina con un solo procesador.

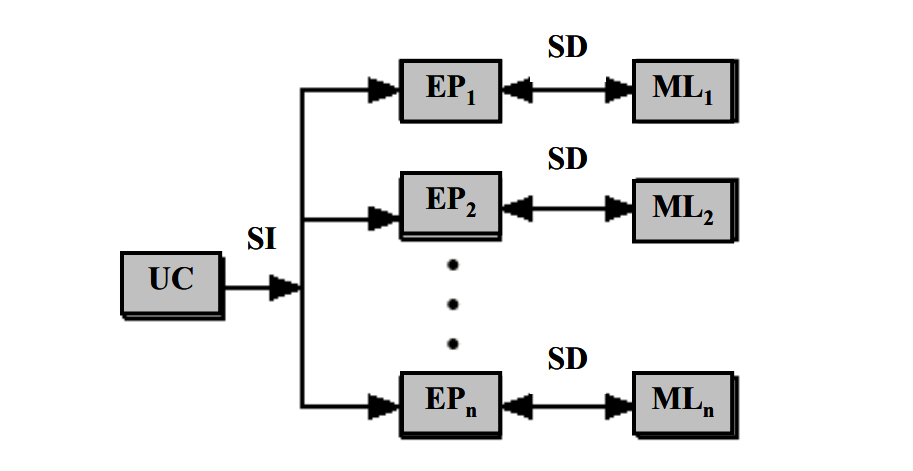
****

**Tipos de sistemas paralelos (categorías de computadoras)**

**SISD**: una secuencia de instrucciones y una secuencia de datos. Un único procesador interpreta una única secuencia de instrucciones (SI) para operar con los datos almacenados en una única memoria(UM) .Computadoras monoprocesador caen en esta categoría.



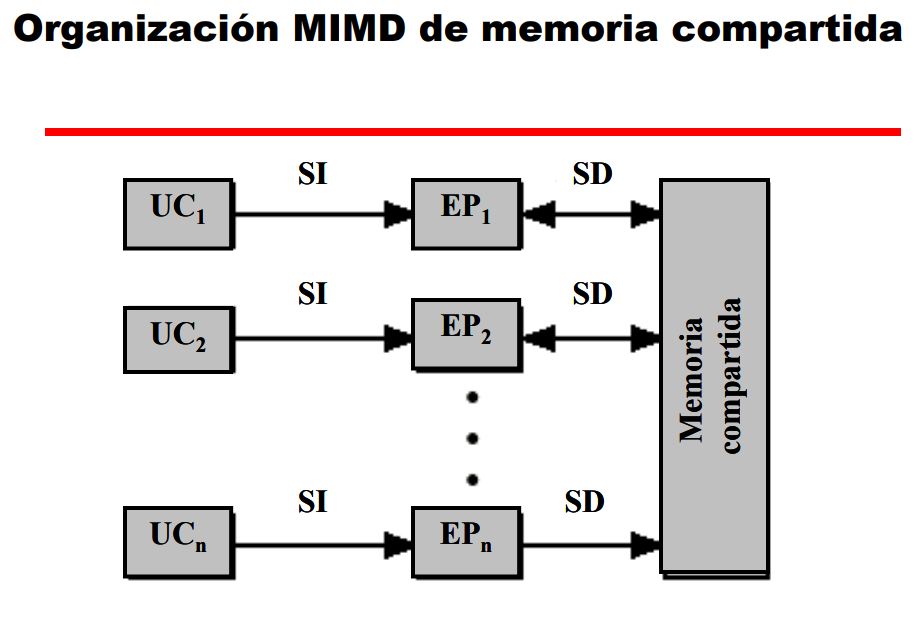
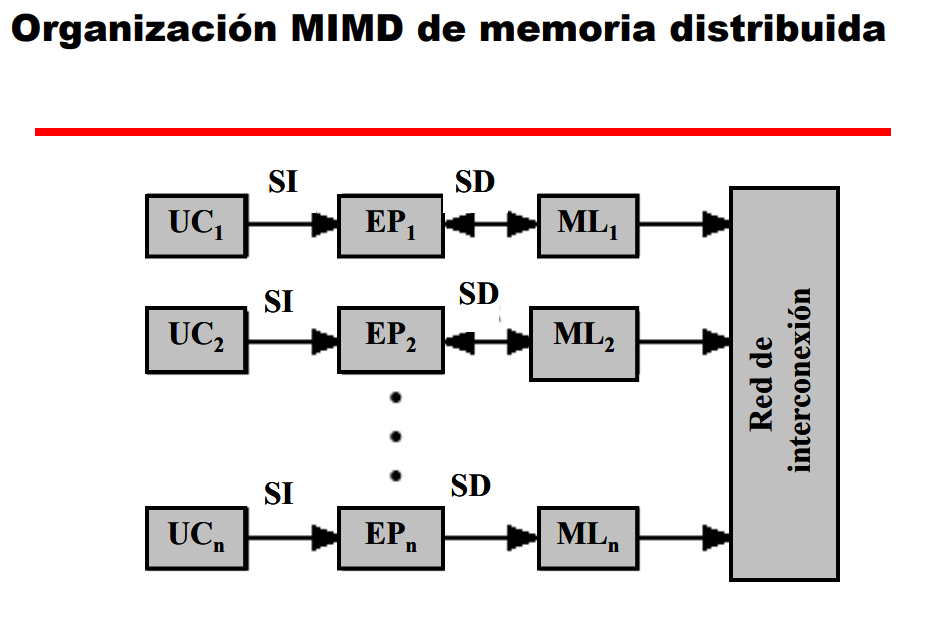
**SIMD:** una secuencia de instrucciones y múltiples secuencias de datos. Una única instrucción máquina controla paso a paso la ejecución simultánea de un cierto número de elementos de proceso (EP). Cada elemento de proceso tiene una memoria dedicada (ML). Procesadores vectoriales y matriciales.



**MISD:** múltiples secuencias de instrucciones y una secuencia de datos. Se transmite una secuencia de datos a un conjunto de procesadores. Cada procesador ejecuta una secuencia de instrucciones diferente. Esta estructura nunca ha sido implementada.

**MIMD:** múltiples secuencias de instrucciones y múltiples secuencias de datos. Un conjunto de procesadores ejecuta secuencias de instrucciones diferentes en simultáneo con conjuntos de datos diferentes. Los SMP, clusters y sistemas paralelos son ejemplo de esta categoría.

Se pueden dividir según la forma de comunicarse

**Multiprocesador simetrico (SMP)**

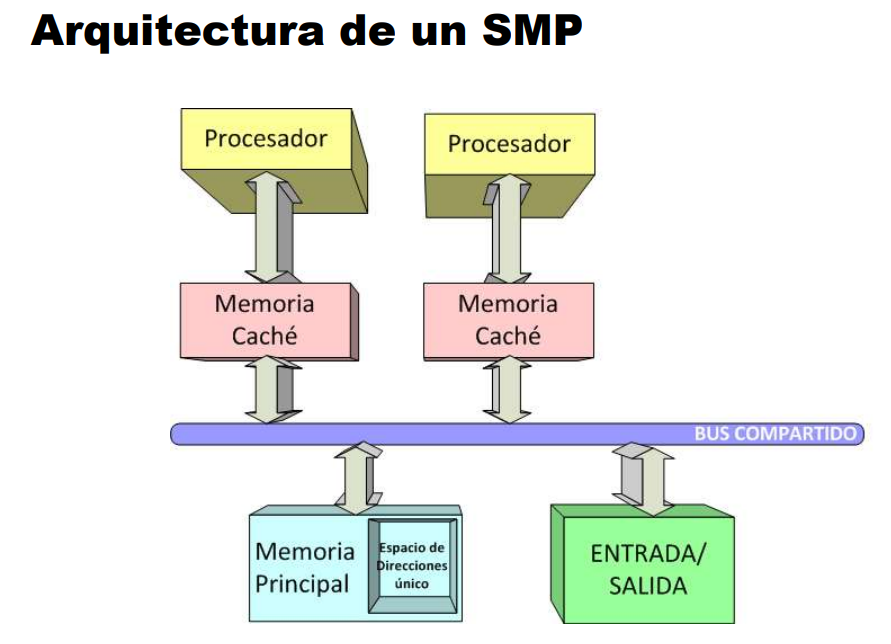
Computadora autónoma con las siguientes características:

• Dos o más procesadores similares interconectados mediante un bus u otro tipo de sistema de interconexión que pueden desempeñar las mismas funciones.

• Comparten la memoria principal y las E/S. El tiempo de acceso a memoria es similar para todos los procesadores (UMA).

Un SMP tiene las siguientes ventajas frente a un monoprocesador:

* Mayores prestaciones: si el trabajo a realizar puede organizarse en paralelo.
* Buena disponibilidad: un fallo en un procesador no detendrá la computadora
* Crecimiento incremental: Se pueden añadir más procesadores y es escalado: En función de la cantidad de procesadores
* Cuidado: Bus compartido



**Clusters**

Computadoras completas interconectadas que trabajan conjuntamente como un único recurso de cómputo, creando la ilusión de que se trata de una única máquina.

Con clusters tenemos 4 beneficios:

* Escabilidad absoluta: Es posible configurar clusters grandes, que pueden superar las prestaciones de las computadoras independientes más potentes.
* Escabilidad incremental: Un cluster se configura de forma que sea posible añadir nuevos sistemas a este en ampliaciones sucesivas.
* Alta disponibilidad: El fallo de uno de los nodos no significa la pérdida del servicio. El software proporciona la tolerancia a fallos.
* Mejor relación precio/prestaciones: Es posible configurar un cluster con mucha potencia a menos costo que un computador independiente.

**Clúster vs SMP**

Ambos: Dan soporte a aplicaciones de alta demanda de recursos y están disponibles comercialmente (SMP es más antiguo).

SMP :Más fácil de administrar y configurar y más cercano a los sistemas de un solo procesador

CLUSTER: Superior escalabilidad incremental absoluta y superior disponibilidad (Redundancia)

**UMA**

Acceso uniforme a memoria: Todos los procesadores pueden acceder a la memoria principal utilizando instrucciones de carga y almacenamiento. El tiempo de acceso es igual a todas las regiones de memoria, así como el tiempo de acceso a memoria es el mismo para todos los diferentes procesadores.

**NUMA**

Acceso no uniforme a memoria: Todos los procesadores tienen acceso a todas las partes de la memoria principal utilizando instrucciones de carga y almacenamiento. El tiempo de acceso a memoria principal de un procesador depende de la región a la que se acceda. Diferentes procesadores acceden a diferentes regiones de memoria principal a diferentes velocidades.

**CC-NUMA**

NUMA con coherencia de cache: Un computador NUMA en el que la coherencia de cache se mantiene en todas las caches de los distintos procesadores.

**Procesamiento multihebra**

La secuencia de instrucciones se divide en secuencias más pequeñas llamadas hebras (threads) que pueden ejecutarse en paralelo. Aumenta el paralelismo de instrucciones sin aumentar la complejidad y consumo de potencia de la segmentación de cauce y los superescaleres. Hay una amplia variedad de diseños multihebra.

**Hebra (thread):** Unidad de trabajo de un proceso que puede asignarse, incluye un contexto de procesador (incluido PC y SP) y área de datos para su pila (stack). Esta se ejecuta secuencialmente y es interrumpible pues el procesador cambiaría a otra hebra.

**Conmutación de hebra (thread switch):** Consiste en el cambio de control del procesador entre hebras de un mismo proceso.

**Multihebra explícito:** Ejecución concurrente de instrucciones de diferentes hebras explicitas. Hay una mezcla de instrucciones de diferentes hebras en cauces compartidos, la ejecución también puede ser paralela en cauces paralelos. Todos los procesadores comerciales lo usan.

**Multihebra implícito:** Ejecución concurrente de varias hebras extraídas de un único programa secuencial, definidas estáticamentepor el compilador o dinámicamente por el hardware.

En un procesador multihebra hay PC (contador progama) distintos para cada hebra que pueda ejecutarse concurrentemente. Se trata cada hebra separadamente, obtenemos paralelismo entre hebras.